University of Illinois Urbana-Champaign

ECE 445 Senior Design

SLV226: Camera-Based Obstacle-Avoidance Car

Final Report

Team

Tianxi Zhu, Yuxuan Liu, Zhuo Li, Zihao Wu

May 2026

Abstract

SLV226 is a compact mobile robot built for a structured indoor obstacle-avoidance demonstration. The prototype combines an OpenMV camera, an STM32F103 microcontroller, wheel encoders, an MPU6050 inertial measurement unit (IMU), an ESP8266 Wi-Fi module, and a browser dashboard connected through Message Queuing Telemetry Transport (MQTT). The OpenMV camera detects AprilTag-marked obstacles and sends relative-pose information to the STM32 through a lightweight Universal Asynchronous Receiver/Transmitter (UART) protocol. The STM32 then uses this information together with wheel-speed and yaw feedback to support straight driving, turning, and automatic obstacle avoidance.

The avoidance behavior is implemented around the existing motion-command queue. When a tagged obstacle enters a configurable danger region in front of the vehicle, the controller inserts a three-step response: turning away from the obstacle, translating for a short distance, and recovering the heading. After the sequence completes, the vehicle can return to forward motion if the original command is still valid. In parallel, the ESP8266 publishes vehicle state and obstacle data to the browser dashboard, where the operator can monitor the system, adjust thresholds, and issue high-level commands.

The current prototype is intended for controlled classroom demonstrations and robotics experiments. Its main limitation is that obstacle detection depends on AprilTag-marked hazards rather than arbitrary objects. Within that scope, the platform demonstrates how perception, embedded control, wireless communication, and operator feedback can be integrated into one small robotic system.

Contents

1 Introduction

1.1 Problem Statement and Motivation

Small mobile robots used in indoor demonstrations often need more than manual commands and open-loop motor timing. In this project, the car needs to keep its motion stable, recognize a prepared obstacle in front of it, and report enough internal state for debugging and supervision. Open-loop driving is not reliable enough by itself because wheel mismatch, floor conditions, and supply variation can cause drift. Manual supervision is also limited because the operator may not always see both the robot's surroundings and its internal state.

SLV226 was developed as a compact obstacle-aware robotic car for this setting. The project combines closed-loop motion control, embedded vision, wireless telemetry, and a browser-based operator interface. The intended behavior is straightforward: the car drives under local feedback control, identifies a tagged hazard in front of it, executes a repeatable avoidance maneuver, and reports its state to the user.

A camera-based approach was selected because visual sensing can provide more spatial information than contact switches or single-point range triggers. Fully general object recognition was outside the scope of the available hardware and project schedule. Instead, the prototype uses AprilTag markers as structured obstacles.[1, 6] This gives the OpenMV camera a reliable target for tag identification and pose estimation, while still giving the STM32 geometric information that can be used for avoidance decisions.

The result should be viewed as a structured obstacle-avoidance prototype, not as a general autonomous navigation system. This narrower scope allows the report to focus on integration, control behavior, communication, and verification under the tested marker-based conditions.

1.2 Project Function and Requirements

The vehicle is organized around six top-level requirements. These requirements describe the behavior expected from the prototype and are used as the basis for the verification plan in Section 3.

R1. The car shall maintain closed-loop translational and turning motion using encoder and inertial feedback rather than open-loop timing alone.

R2. The perception subsystem shall detect a tagged obstacle and estimate its relative position with respect to the vehicle.

R3. The embedded controller shall determine whether a detected obstacle lies inside a configurable danger region in front of the car.

R4. The controller shall execute a repeatable avoidance maneuver and then restore forward motion after the maneuver finishes.

R5. The system shall publish robot status and obstacle information wirelessly so that an operator can monitor performance in real time.

R6. The operator shall be able to tune key avoidance parameters from a browser interface without reflashing the firmware.

Several constraints shaped these requirements. The STM32F103-class controller has limited memory and computing resources, so the design avoids heavy image inference and large map structures. The prototype is assembled from low-cost development modules, which makes integration easier but

also introduces packaging, wiring, and calibration tradeoffs. The intended use case is a supervised classroom demonstration, so a marker-based perception method is acceptable as long as the limitation is stated clearly.

The wireless dashboard is used for monitoring and parameter tuning, but it is not treated as the real-time controller. Obstacle checking, warning output, and the avoidance maneuver remain on the STM32 so that the most time-sensitive behavior does not depend directly on the wireless link.

1.3 System-Level Overview

SLV226 is organized into four cooperating blocks: vision, embedded control, wireless communication, and the operator dashboard. Figure 1 gives a hardware-level overview of these blocks and their main connections.

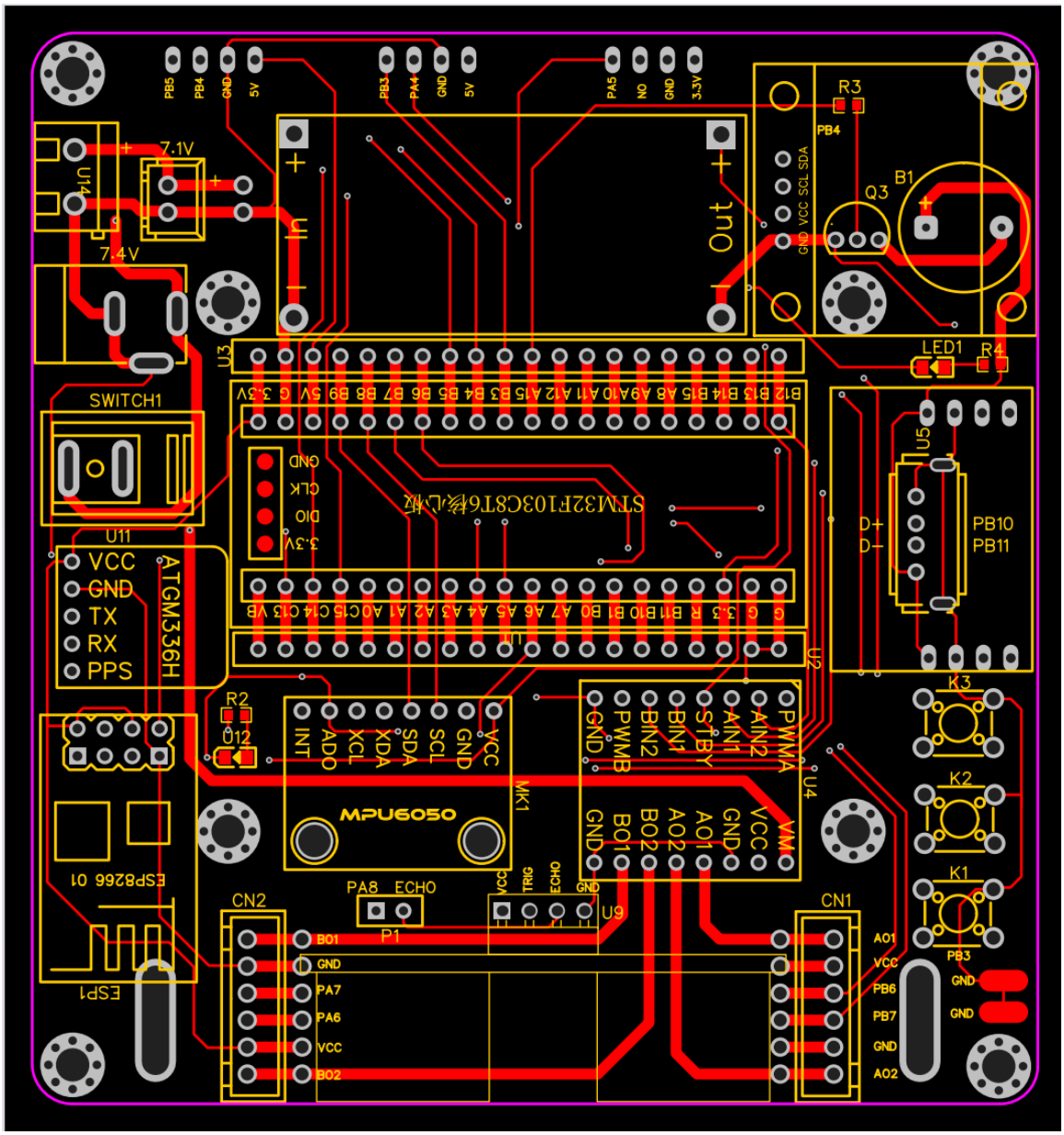

Figure 1: Hardware-level system overview of the SLV226 camera-based obstacle-avoidance car.

The vision block is implemented on the OpenMV camera. It captures grayscale images, detects AprilTag markers, estimates each tag's pose relative to the camera, and sends a compact obstacle frame over UART. Sending geometric data instead of raw images keeps the communication load low and gives the STM32 data that can be used directly by the avoidance logic.

The embedded control block is centered on an STM32F103 microcontroller.[2] It receives obstacle frames, stores active obstacle data, estimates wheel speed and heading from encoder and IMU feedback, and drives the motors through proportional-integral-derivative (PID) control. The same controller checks whether the current motion state allows avoidance and whether any observed obstacle lies inside the configured danger region. If the trigger condition is met, the controller starts the avoidance sequence and later restores forward motion when the sequence is complete.

The wireless communication block uses an ESP8266-01 module connected to the STM32 over USART2. In the current prototype, the ESP8266 joins a local Wi-Fi access point, opens a TCP connection to the Bemfa MQTT service, enters transparent-transmission mode, and carries MQTT packets between the STM32 and the cloud endpoint. This lets the STM32 publish telemetry, obstacle packets, and acknowledgments while also receiving browser commands.

The dashboard is the user-facing part of the system. It shows connection status, robot state, obstacle information, and avoidance parameters. It also lets the operator enable or disable avoidance, start motion, and adjust thresholds. In a typical run, the camera detects a tagged obstacle, the STM32 parses and evaluates the packet, the controller executes avoidance if needed, and the dashboard records the obstacle condition and vehicle response.

1.4 Expected User Workflow

A typical test starts by powering the vehicle and keeping it stationary while the IMU calibrates. During initialization, the controller starts the OLED display, motor subsystem, encoder interfaces, IMU, PID structures, ESP8266 Wi-Fi path, and camera receive path. The 10 ms control interrupt is enabled only after these subsystems are ready. On the local display, the ESP8266 setup moves through AT test, module reset, station-mode configuration, Wi-Fi association, single-connection configuration, TCP connection, transparent-transmission configuration, and transmit enable. After the MQTT handshake succeeds and the OLED reports that the connection is ready, the operator opens the browser dashboard and checks that telemetry is updating.

The operator then sets the lateral threshold, range threshold, avoidance turn angle, and avoidance travel distance. After these parameters are configured, the operator sends an avoidance-enable command and then issues a forward manual-motion command from the dashboard. During motion, the controller maintains heading using encoder and gyroscope feedback. If the camera detects a tagged obstacle inside the danger region, the dashboard logs the event, the local alarm becomes active when appropriate, and the car executes the avoidance maneuver. After the maneuver finishes, the car returns to forward motion automatically if the pre-avoidance manual-forward state is still valid.

1.5 Project Scope and Success Criteria

The current version of SLV226 is intended to detect tagged hazards, execute a consistent avoidance response, communicate that response to the operator, and restore forward motion afterward. It is not intended to recognize arbitrary unmarked obstacles under all environmental conditions. Within that scope, the system can be considered successful if it repeatedly performs the intended sequence with acceptable consistency: stable forward motion, correct hazard identification inside the configured danger region, reliable avoidance execution, and clear operator feedback.

1.6 Broader Impacts

The main broader value of SLV226 is as an educational and debugging platform. It shows how sensing, control, communication, firmware structure, and user-interface design have to work together in a small robotic system. The dashboard and local indicators also make the prototype easier to inspect: obstacle data, queue state, connection state, and warning behavior can be checked during a run instead of inferred afterward.

The project also has a scope-related responsibility. A marker-based avoidance system should not be

presented as a general safety system for autonomous navigation. Its strengths and limits should be stated clearly whenever the project is demonstrated. Within those limits, SLV226 is a useful example of an integrated robotics prototype built around observable behavior and controlled test conditions.

2 Design

2.1 Design Procedure

The design work was divided into perception, embedded control, communication, and human interface blocks. For each block, the team chose an approach that could be implemented on the available hardware while keeping the system testable.

For perception, the main options were simple range sensing, general object detection, and marker-based visual detection. Simple range sensors would have reduced image-processing work, but they would not provide obstacle identity or pose. General object detection would be more flexible, but it would require more computation and training effort than was practical for the STM32 and OpenMV hardware. Marker-based detection was selected because AprilTag recognition gives a structured target and provides relative-pose information in the indoor test environment.

For control, the main options were open-loop timing, computer-side control, and embedded closed-loop control. Open-loop timing was not sufficient because wheel mismatch, floor friction, and battery variation can cause drift. Computer-side control was avoided because wireless delay would then affect motor output directly. Embedded closed-loop control was selected so that the STM32 could generate motor commands locally while using encoder and IMU feedback.

For communication, the main options were direct serial operation, Bluetooth serial, and Wi-Fi/MQTT telemetry. Direct serial operation is useful for debugging, but it keeps the vehicle tethered to a host computer. Bluetooth serial is simple, but it is less convenient for a browser-based dashboard. Wi-Fi with MQTT was selected because it separates commands and telemetry through a publish-subscribe structure and fits the dashboard workflow.[7] Real-time control still remains on the vehicle so that network delay does not directly determine motor behavior.

For the human interface, the main options were a local-only display, a computer serial monitor, and a browser dashboard. A local display is useful but too small for obstacle records and parameter tuning. A serial monitor is useful during development but awkward during a demonstration. The browser dashboard was selected because it can show telemetry, obstacle state, command acknowledgments, and adjustable avoidance parameters in one place. The OLED, LED, and buzzer remain as local status indicators.

For the power system, the design emphasized safe switching, stable logic supply, and separation between motor power and logic power where practical. The motor path can draw transient current during startup and turning, so it was kept away from sensitive logic and communication paths while still sharing a common ground reference.

2.2 Hardware Architecture

The hardware system is organized around the STM32F103C8T6 microcontroller, which serves as the vehicle controller. The STM32 coordinates motor control, sensor feedback, camera communication, wireless communication, and local user-interface functions. The modular structure made it possible to test individual subsystems before combining them into the full vehicle.

Figure 2 shows the printed-circuit-board layout. Table 1 summarizes the main hardware modules used in the vehicle and the function of each subsystem.

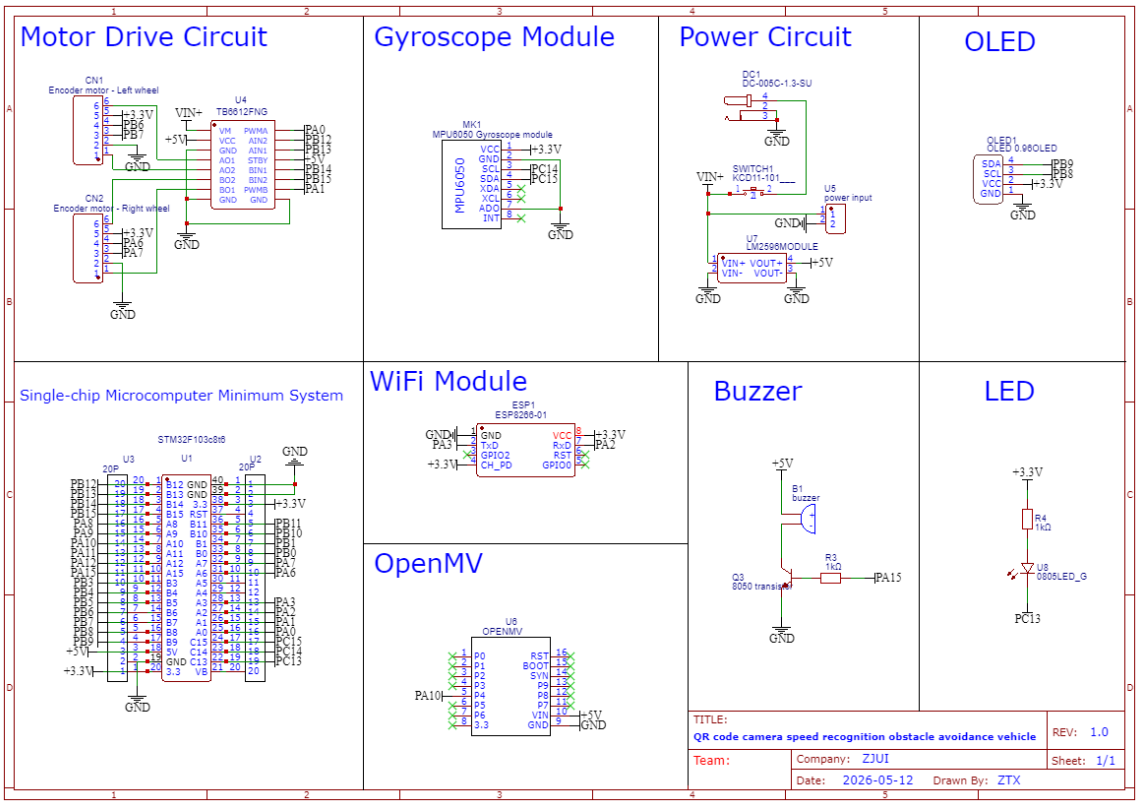

Figure 2: Printed-circuit-board layout for the SLV226 vehicle control system.

Table 1: Main hardware modules and functions.

Subsystem	Main part	Function
Main controller	STM32F103C8T6	Coordinates motor control, sensor reading, camera communication, wireless communication, display update, and overall decision logic.
Motor driver	TB6612FNG	Drives the left and right dc motors using PWM and direction-control signals from the STM32.
Motion sensing	Encoder motors and MPU6050	Encoder signals provide wheel-speed feedback, while the MPU6050 provides inertial motion information for heading support.
Wireless module	ESP8266-01	Provides Wi-Fi communication between the vehicle and the browser dashboard.
Vision module	OpenMV camera	Performs AprilTag recognition and sends processed recognition results to the STM32.
Human interface	OLED, LED, buzzer	Displays local status information and provides visual or audio indication during testing and operation.
Power path	DC input, switch, regulator	Distributes external power and generates regulated supply rails for the motors, controller, and peripheral modules.

The motion-control subsystem uses a TB6612FNG dual-channel motor driver to control the left and right dc motors. The STM32 sends PWM and direction-control signals to the motor driver, and the driver supplies the current required by the motors. This is necessary because microcontroller input/output pins cannot drive the motors directly. Encoder signals from the motors return to the STM32 so that the controller can estimate wheel speed and monitor the vehicle's motion state.

The vision subsystem uses an OpenMV camera module. Instead of sending raw image data to the STM32, the OpenMV module processes the image locally and sends simplified recognition results to the main controller. This reduces the workload on the STM32 and leaves the microcontroller focused on motor output, encoder processing, and obstacle-response logic.

The sensing and interface subsystems provide additional feedback for control and debugging. The MPU6050 IMU provides motion-related sensing information. The OLED display shows local system status, while the LED and buzzer provide simple visual and audio indications. The ESP8266-01 Wi-Fi module supports wireless data exchange between the vehicle and the browser dashboard.

At the signal level, the current firmware uses USART1 at 115200 bps for camera data and USART2 at 115200 bps for ESP8266 communication. The camera data path uses PA10 as the receive pin on the STM32, while the Wi-Fi path uses PA2 and PA3 for ESP8266 transmission and reception. The local warning indicators are driven through PA15 for the buzzer and PC13 for the LED, with JTAG disabled so that PA15 can be reused as a general-purpose output. These assignments match the firmware initialization sequence, which enables the required interrupt-driven receive paths and GPIO remapping before normal operation begins.

2.3 Power System and PCB Layout Considerations

The power system distributes the external dc input to both the motor-driving path and the low-voltage logic modules. The input first passes through a mechanical power switch so the vehicle can be turned on and off safely during testing. A buck regulator module generates regulated rails for the controller and peripheral modules. In the schematic, the power block includes the dc input connector, switch, regulator module, and the 5 V and 3.3 V supply nets.

Motor noise was one of the main layout concerns. During startup, acceleration, sudden direction changes, or wheel blocking, the dc motors can draw transient current. These current spikes may disturb the supply voltage and cause the STM32, camera module, or Wi-Fi module to reset. For that reason, the motor power path and logic power path were kept as separate as practical while all modules still share a common ground reference. The common ground is needed because the control and communication signals must use the same voltage reference.

The PCB and wiring layout should keep high-current motor traces away from sensitive communication and sensor lines. The motor driver should be close to the motor connectors to reduce the length of the high-current path. Decoupling capacitors for the microcontroller and communication modules should be placed near their power pins. UART and inter-integrated circuit (I2C) signal lines should be kept short and routed away from the motor outputs when possible. These layout choices are intended to reduce voltage fluctuation, communication errors, and unstable behavior during vehicle motion.

2.4 Wireless Communication and Human Interface

The wireless communication system uses the ESP8266-01 module to connect the vehicle to the browser dashboard through Wi-Fi and MQTT. In the implemented firmware, the ESP8266 associates with a configured access point, opens a TCP connection to `bemfa.com:9501`, enables transparent mode, and exchanges MQTT packets on behalf of the STM32. The browser dashboard connects to the same service through secure WebSocket transport at `wss://bemfa.com:9504/wss`. Commands are sent from the dashboard to the vehicle, while telemetry messages are sent from the vehicle back to the dashboard. Keeping the two directions separate makes the message flow easier to debug.

The dashboard sends high-level commands such as motion commands, avoidance-enable signals, and parameter updates. In the current implementation, the browser publishes command strings such as `MANU,F,...`, `STOP`, `AVOID,1`, and `AVPARM,...` to topic `slv226cmd`. The ESP8266 receives these messages and forwards them to the STM32 through its interrupt-driven UART receive path. The STM32 then updates its internal control state and executes the corresponding local behavior. In the opposite direction, the STM32 sends vehicle status, obstacle information, acknowledgments, and avoidance-event messages to the ESP8266, which publishes them on topic `slv226data`.

Real-time motor control is not performed directly by the browser dashboard. The STM32 generates PWM and direction-control signals locally, which reduces the effect of wireless delay on vehicle motion. If the wireless connection becomes delayed or temporarily unavailable, the local controller can still execute local behaviors such as stopping or obstacle avoidance.

Table 2 summarizes the command and telemetry messages used by the wireless communication interface. Figure 3 shows the browser dashboard.

Table 2: Command and telemetry summary.

Topic or message	Direction	Purpose
`slv226data`	Vehicle to dashboard	Publishes telemetry data such as vehicle state, speed, obstacle status, or recognition result.
`slv226cmd`	Dashboard to vehicle	Sends operator commands such as motion control, mode switching, or parameter updates.
`SPD:...` `DIS:...` `YAW:...`	Vehicle to dashboard	Reports periodic telemetry including wheel speed, displacement, yaw, queue state, and current motion mode.
`OBS:...`	Vehicle to dashboard	Reports obstacle-related information detected by the vehicle.
`AVD:HIT` and `AVD:DONE`	Vehicle to dashboard	Indicates that obstacle avoidance has been triggered or completed.
`MANU,F,...` and `STOP`	Dashboard to vehicle	Starts forward manual motion with heading hold or immediately halts motion.
`AVOID,1/0`	Dashboard to vehicle	Enables or disables the obstacle-avoidance function.
`AVPARM,...`	Dashboard to vehicle	Updates avoidance-related parameters such as threshold, timing, or response settings.

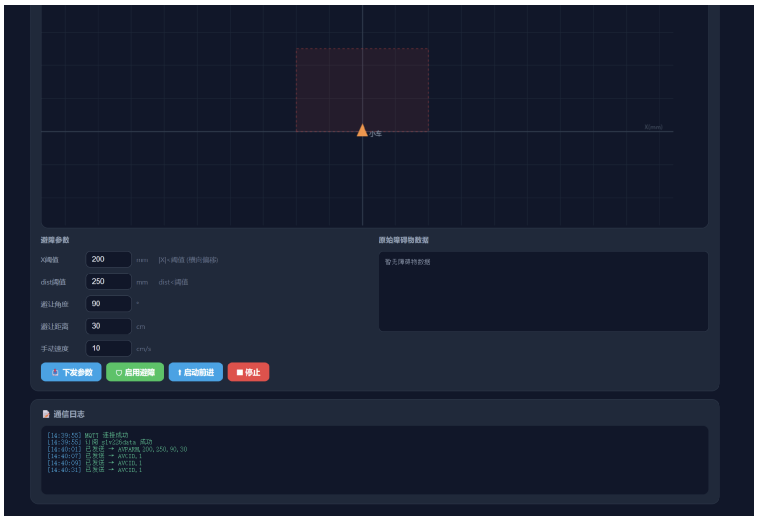

Figure 3: Browser dashboard screenshot.

The browser dashboard is used during testing and demonstration. The operator can send commands, enable or disable obstacle avoidance, adjust selected parameters, and observe telemetry data reported by the vehicle. In the current workflow, the operator opens the dashboard, confirms topic connectivity, sends `AVPARM` to update thresholds, sends `AVOID,1` to enable avoidance, and then sends `MANU,F,...` to begin forward motion. This avoids having to connect the STM32 directly to a host

computer for every test.

The onboard human interface complements the dashboard. The OLED display can show local information such as operating mode, connection status, speed, or recognition result. The LED and buzzer provide simple visual and audio feedback for startup, command reception, obstacle detection, or warning conditions. These local indicators are useful during debugging because they show whether the vehicle is operating even when the wireless dashboard is unavailable.

2.5 Vision Processing and Obstacle Representation

The vision module reduces each camera image to a small set of obstacle records. Each record contains an obstacle identifier, relative position, estimated distance, and orientation data. The STM32 does not need raw image frames for the current avoidance logic, so the compact record format is sufficient.

For a pinhole camera model, the relationship between a three-dimensional point (X, Y, Z) in the camera coordinate frame and the corresponding image pixel (u, v) can be written as

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}, \tag{1}$$

where f_x and f_y are focal lengths in pixels, and (c_x, c_y) is the principal point. Equation 1 gives the geometric basis for pose estimation. In the implemented system, the OpenMV camera performs marker detection and pose estimation locally, then forwards only compact obstacle records to the STM32.[6, 1]

The STM32 receives each obstacle frame through UART. A valid frame begins with a two-byte header, contains a fixed-length payload, includes a checksum, and ends with a tail byte. This framing lets the parser reject corrupted packets and recover after random bytes. The obstacle table stores up to three active obstacles. Repeated obstacle identifiers refresh existing entries, a fourth active obstacle replaces the oldest entry, and stale entries are invalidated after 500 ms without a refresh.

2.6 Motion Control and Avoidance Algorithm

The vehicle uses separate wheel-speed loops for the left and right motors. For each motor, the discrete PID controller is expressed as

$$u[k] = K_p e[k] + K_i T \sum_{i=0}^{k} e[i] + K_d \frac{e[k] - e[k-1]}{T}, \tag{2}$$

where $e[k]$ is the error signal at time step k, T is the sampling period, and $u[k]$ is the controller output. The default prototype gains were $K_p = 1.0$, $K_i = 0.25$, and $K_d = 0.02$. These gains were adequate for low-speed demonstration testing, but they should be re-tuned if the vehicle mass, battery voltage, wheel size, or driving surface changes.

Obstacle avoidance is based on a configurable trigger region in front of the car. For obstacle i, let X_i be the lateral offset from the vehicle centerline and d_i be the estimated obstacle distance. The obstacle is considered to be in the danger region when

$$|X_i| < X_{\text{th}} \quad \text{and} \quad d_i < d_{\text{th}}, \tag{3}$$

where X_{th} is the lateral threshold and d_{th} is the distance threshold. The thresholds can be changed from the browser dashboard during testing.

When the condition in Equation 3 is satisfied during an eligible forward-driving state, the controller inserts an avoidance sequence into the motion-command queue. The sequence consists of a turn away from the obstacle, a short translation, and a heading-recovery command. After the sequence finishes, the controller resumes forward motion if avoidance remains enabled and no stop command has been issued. Reusing the same motion primitives for both normal motion and avoidance keeps the firmware simpler and reduces special-case logic.

3 Verification

3.1 Verification Strategy

The verification work checked whether the completed vehicle satisfied the major functional requirements under demonstration conditions. The main items were camera packet reliability, obstacle data freshness, web telemetry, speed-control repeatability, warning output, and the end-to-end obstacle-avoidance response.

The tests were divided into several categories. First, the camera interface was tested to confirm that the STM32 accepted valid packets and rejected corrupted packets. Second, the obstacle storage logic was checked for refresh, replacement, and timeout behavior. Third, the telemetry and web interface were tested through MQTT messages and browser visualization. Fourth, the motor speed-control system was checked during low-speed motion. Finally, the integrated avoidance behavior was tested by placing a target inside the programmed danger zone and observing the response sequence.

The top-level verification test required the following event sequence:

1. The camera transmits a valid obstacle frame.
2. The STM32 stores the obstacle as valid.
3. The obstacle satisfies $|X| < X_{\text{th}}$ and $d < d_{\text{th}}$.
4. The vehicle interrupts normal forward motion.
5. The STM32 publishes `AVD:HIT`.
6. The avoidance path executes.
7. The STM32 publishes `AVD:DONE`.

This strategy checks the system as an integrated loop instead of treating every module as an isolated part. The full requirement-and-verification table is provided in Appendix A.

3.2 Current Verification Results

The current prototype passed the functional checks used for the controlled demonstration. Camera frames were parsed safely, stale obstacles were removed after timeout, MQTT telemetry updated the dashboard, low-speed motor behavior was repeatable, warning outputs worked, and the car executed the expected avoidance sequence when a tagged target entered the danger zone.

The camera module reports obstacle information to the STM32 through USART1. Each valid frame begins with `0xAA 0x55`, contains a 16-byte payload, includes an exclusive-or checksum over the payload, and ends with `0x5A`. Test inputs included one correct packet, one packet with a wrong header, one packet with a wrong checksum, and one packet with a missing tail byte. Only the correct frame updated the obstacle table, while corrupted frames were rejected.

The firmware stores up to three valid obstacles. This was verified by sending obstacle identifiers 1, 2, and 3, then sending a fourth identifier to confirm that the oldest slot was replaced. Repeated obstacle identifiers refreshed the existing slot instead of creating a duplicate. The aging requirement was also verified: stale obstacle entries were invalidated after 500 ms without a refresh.

The ESP8266 and browser dashboard provide remote visibility into the vehicle state. During startup, the ESP8266 completed Wi-Fi association, TCP connection, transparent-transmission entry, and MQTT login before the operator interface was used. The STM32 publishes obstacle data through

MQTT using the format

`OBS:id,X,Y,Z,dist,rx,ry,rz;...`

or

`OBS:NONE`

when no valid obstacle exists. Because the firmware uses a 200 ms publish interval, the telemetry update rate is approximately 5 Hz during demonstration testing.

The telemetry path also publishes a compact periodic status record containing wheel speed, displacement, yaw, queue depth, motion-state code, and yaw-correction information on topic `slv226data`. The dashboard parsed both the status format and obstacle messages while also sending control commands on topic `slv226cmd`. The firmware also implements MQTT keep-alive and reconnect behavior so that the controller can recover from a dropped wireless session without reflashing or rebooting the vehicle.

The vehicle uses encoder feedback and separate PID loops for left and right wheel speed. The default PID gains in Equation 2 were sufficient for the low-speed avoidance sequence used in the demonstration. Higher-speed behavior and different floor conditions were not fully characterized in this report.

3.3 Remaining Measurements

Several measurements remain before making stronger performance claims. Camera calibration should be checked with known target positions so that pixel-to-position error can be reported quantitatively. Tag-detection distance should be measured under the lighting conditions used in the final demonstration. Avoidance success rate should be measured over repeated runs rather than described only from a demonstration sequence. Heading recovery should be measured as final yaw error after the avoidance maneuver. Table 3 summarizes these items.

Table 3: Verification items to complete or strengthen.

Test item	**What to measure**	**Status**
Camera calibration quality	Pixel-to-position error and calibration repeatability.	Missing systematic measurement.
Reliable tag-detection distance	Maximum stable detection distance under normal lighting.	Partially tested.
Straight-drive performance	Left/right wheel balance and path deviation at low speed.	Passed for prototype.
Avoidance trigger region	Boundary where $\lvert X\rvert < X_{\text{th}}$ and $d < d_{\text{th}}$ activates avoidance.	Passed for demonstration.
Avoidance success rate	Number of successful avoidance trials over repeated runs.	Missing repeated-trial data.
Heading recovery after avoidance	Final heading error after the avoidance maneuver.	Missing quantitative measurement.
Alarm behavior	LED and buzzer response under warning condition.	Passed.
MQTT responsiveness	Message update rate and command-acknowledgment delay.	Passed for demonstration.

4 Costs

The cost estimate includes the prototype hardware used to reproduce the vehicle. The retail estimate reflects approximate purchased-part cost, while the paid estimate is lower where parts were available from lab stock or reused from previous work. The largest non-labor cost items are the camera module, motors with encoders, chassis, battery, and custom PCB. If the design were produced in larger quantities, the STM32 controller, motor driver, connectors, and passive components would likely become cheaper through bulk purchasing and integration. The camera and mechanical system would remain important cost targets because they affect both perception performance and vehicle durability.

Table 4: Prototype parts-cost worksheet.

Item	**Qty.**	**Retail cost**	**Paid cost**
OpenMV camera module / AprilTag camera module	1	$35.00	$35.00
STM32F103C8T6 board	1	$6.00	$6.00
TB6612FNG module	1	$5.00	$5.00
MPU6050 module	1	$3.00	$3.00
ESP8266-01 module	1	$4.00	$4.00
OLED module	1	$5.00	$5.00
Power module and switch	1 set	$18.00	$13.00
Encoder motors and chassis parts	1 set	$42.00	$30.00
PCB and miscellaneous hardware	1 set	$43.00	$29.00
Total		**$161.00**	**$130.00**

5 Conclusions

The final prototype brings the main subsystems into one working loop. The camera detects tagged obstacles and sends relative coordinates to the STM32 over USART1. The STM32 checks the packet structure, stores fresh obstacle data, controls the motors with encoder feedback, reports telemetry through MQTT by way of the ESP8266 USART2 link, and starts the avoidance path when an obstacle enters the programmed danger zone. The browser dashboard gives the operator visibility into the obstacle state and allows avoidance parameters to be adjusted during testing.

The strongest result is the end-to-end demonstration: camera detection leads to an STM32 decision, a motor-control response, and dashboard event messages. The requirement-and-verification table also records the functional checks that were completed, so the report is not based only on a visual demonstration.

The main limitations are tied to the project scope. The camera-based approach supports avoidance only when targets are visible and correctly interpreted. The firmware stores three obstacle entries, which is enough for the controlled demonstration but not enough for cluttered environments. The MQTT connection used in testing is convenient but not secure enough for deployment. Future work should add authenticated communication, stronger camera calibration, a physical emergency stop, longer battery testing, and more systematic obstacle-priority logic.

6 Ethical Considerations

This project should be presented as a constrained prototype, not as a fully autonomous vehicle. The vehicle should be described as a camera-assisted obstacle-avoidance prototype under tested visual-marker conditions. It should not be operated near people, stairs, roads, pets, fragile objects, or crowded public areas without supervision and additional safety mechanisms.

The communication system also creates safety and privacy concerns. A public MQTT broker is acceptable for a classroom demonstration, but it can expose telemetry and command topics to outside users. A deployment-oriented version would need a private broker, authentication, protected topics, and command authorization. The vehicle should also include a physical emergency stop or motor-power cutoff so that network or firmware failure cannot leave the vehicle moving.

Environmental impact is modest but still relevant. The use of a rechargeable battery and reusable modules reduces waste compared with a disposable design. However, lithium batteries must be charged, stored, and disposed of safely. The project documentation should also make the hardware reusable by future teams so that the PCB, motors, sensors, and battery are not discarded after one semester.

The ethical position of the project is therefore based on clear limits, supervised testing, and honest reporting of what the prototype can and cannot do.

References

[1] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 3400–3407, doi: https://doi.org/10.1109/ICRA.2011.5979561.

[2] STMicroelectronics, "STM32F103x8 STM32F103xB medium-density performance line Arm-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 communication interfaces," datasheet. Available: https://www.st.com/resource/en/datasheet/stm32f103vb.pdf.

[3] Toshiba Electronic Devices & Storage Corporation, "TB6612FNG brush motor driver IC," product page and datasheet. Available: https://toshiba.semicon-storage.com/eu/semiconductor/product/general-purpose-logic-ics/detail.TB6612FNG.html.

[4] TDK InvenSense, "MPU-6000/MPU-6050 product specification," datasheet. Available: https://product.tdk.com/system/files/dam/doc/product/sensor/mortion-inertial/imu/data_sheet/mpu-6000-datasheet1.pdf.

[5] Espressif Systems, "ESP8266EX datasheet," datasheet. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[6] OpenMV, "OpenMV MicroPython documentation: image machine-vision library and AprilTag support," documentation. Available: https://docs.openmv.io/library/omv.image.html?highlight=apriltags.

[7] OASIS, "MQTT Version 3.1.1," standard. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

A Requirement and Verification Table

Table 5: Requirement and verification table.

Requirement	Verification method	Result
Camera UART accepts only valid framed packets.	Send a valid packet, a wrong-header packet, a wrong-checksum packet, and a wrong-tail packet.	Passed. Only the valid packet updated obstacle data.
Parser recovers after corrupted bytes.	Send random bytes before a valid frame.	Passed. Parser resynchronized.
Obstacle table stores three active obstacles.	Send obstacle identifiers 1, 2, and 3.	Passed. Three slots were maintained.
Repeated obstacle identifier refreshes the same slot.	Send the same identifier with changed coordinates.	Passed. Existing slot updated.
Fourth obstacle replaces oldest active slot.	Fill three slots, then send a fourth identifier.	Passed. Oldest slot replaced.
Obstacle ages out after 500 ms without refresh.	Stop sending packets after one valid obstacle frame.	Passed. Stale entry invalidated.
MQTT publishes obstacle status near 5 Hz.	Observe `OBS:` messages over a 5 s interval.	Passed for demonstration.
Dashboard parses `OBS:NONE`.	Publish an empty obstacle message.	Passed. Obstacle count became zero.
Dashboard parses multiple obstacle records.	Publish semicolon-separated records.	Passed. Multiple obstacles rendered.
Avoidance enable command reaches STM32.	Send `AVOID,1` and check acknowledgment.	Passed. Acknowledgment received.
Avoidance parameters update thresholds.	Send `AVPARM` command and retest trigger boundary.	Passed for demonstration.
Vehicle avoids obstacle inside danger zone.	Drive forward with an obstacle satisfying $\lvert X\rvert < X_{\mathrm{th}}$ and $d < d_{\mathrm{th}}$.	Passed. `AVD:HIT` and avoidance path occurred.
Avoidance does not trigger while disabled.	Repeat the danger-zone test with avoidance disabled.	Passed. No avoidance sequence occurred.
Stop command halts motion.	Send `STOP` during forward motion.	Passed. Motor target returned to zero.
Buzzer works on PA15 after JTAG remap.	Disable JTAG, initialize PA15, and toggle warning output.	Passed. Buzzer responded.
LED warning indicates close obstacle state.	Place obstacle inside warning threshold.	Passed. LED state changed.
Wheel-speed response is repeatable at low speed.	Command low manual speed and observe left/right response.	Passed for prototype.

B Supplementary Figures and Cost Details

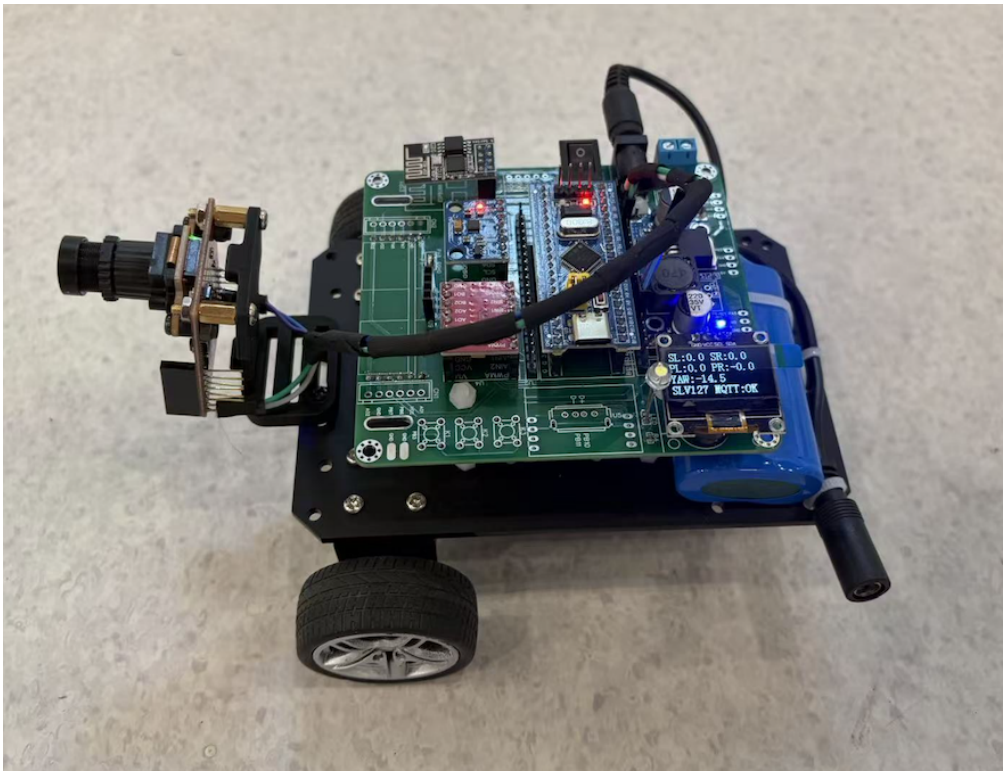

Figure 4: Assembled car photo.

Table 6: Detailed prototype cost table.

Item	Qty.	Retail cost	Paid estimate
STM32F103C8T6 core board	1	$6.00	$6.00
TB6612FNG motor driver	1	$5.00	$5.00
MPU6050 module	1	$3.00	$3.00
ESP8266-01 Wi-Fi module	1	$4.00	$4.00
OLED display	1	$5.00	$5.00
QR/AprilTag camera module	1	$35.00	$35.00
DC gear motors with encoders	2	$24.00	$18.00
Wheel/chassis set	1	$18.00	$12.00
Lithium battery pack	1	$15.00	$10.00
Voltage regulator/power module	1	$3.00	$3.00
Buzzer, LED, resistors, switches, headers	1 lot	$8.00	$4.00
PCB fabrication allowance	1	$25.00	$20.00
Assembly/rework consumables	1 lot	$10.00	$5.00
Parts subtotal		**$161.00**	**$130.00**